View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by K-State Research Exchange

A TOOL FOR IMPLEMENTING DISTRIBUTED ALGORITHMS WRITTEN IN PROMELA, USING DAJ TOOLKIT

by

KRANTHI KIRAN NUTHI

B.Tech, JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, INDIA, 2007

A REPORT

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2009

Approved by:

Major Professor
Dr. Gurdip Singh

Abstract

PROMELA stands for Protocol Meta Language. It is a modeling language for developing distributed systems. It allows for the dynamic creation of concurrent processes which can communicate through message channels. DAJ stands for Distributed Algorithms in Java. It is a Java toolkit for designing, implementing, simulating, and visualizing distributed algorithms. The toolkit consists of Java class library with a simple programming interface that allows development of distributed algorithms based on a message passing model. It also provides a visualization environment where the protocol execution can be paused, performed step by step, and restarted.

This project is a Java application designed to translate a model written in Promela into a model using the Java class library provided by DAJ and simulate it using DAJ. Even though there are similarities between the programming constructs of Promela and DAJ, the programming interface supported by DAJ is smaller, so the input has been confined to a variant, which is a subset of Promela. The implementation was performed in three steps. In the first step an input domain was defined and an ANTLR grammar was defined for the input structure. Java code has been embedded to this ANTLR grammar so that it can parse the input and translates it into an intermediate xml format. In the second step, a String Template is used which would consist of templates of the output model, along with a Java program which traverses the intermediate xml file and generates the output model. In the third step, the obtained output model is compiled and then simulated and visualized using DAJ. The application has been tested over input models having different topologies, process nodes, messages, and variables and covering most of the input domain.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my Major Professor Dr. Gurdip Singh for his constant help, encouragement and guidance throughout the project.

I would also like to thank Dr. Torben Amtoft and Dr. Mitchell Neilsen for serving in my committee and for their valuable cooperation during the project.

Finally, I wish to thank my family and friends for all their support and encouragement.

CHAPTER 1 - INTRODUCTION

Understanding distributed algorithms can become easy when their dynamic behavior can be visualized. There are many programming languages and tools available which can be used to design, simulate and visualize distributed algorithms. Promela is one such programming language which can be used to design distributed algorithms and can be simulated and visualized using SPIN tool. Similarly, DAJ is a Java toolkit which provides a Java class library using which distributed algorithms can be designed and it also provides a visualization environment using Java Abstract Window Toolkit (AWT). This project's main goal is to simulate and visualize a Promela model using DAJ toolkit. In order to do that the given input Promela model has to be translated in to a model which uses the Java class library of DAJ. The following sections of this chapter describe the different technologies used in this project.

1.1 Promela

Promela stands for Protocol Meta Language which is a verification modeling language. It allows for the dynamic creation of concurrent processes. A Promela program mainly comprises of processes, message channels, and variables [2]. Processes represent the concurrent entities of the distributed system. Message channels and variables can be declared either globally or locally within a process. Processes specify behavior, whereas channels and global variables define the environment in which the processes run. The communication between processes is through message channels can be defined to be synchronous or asynchronous. These models can be analyzed with the SPIN model checker to verify that the modeled system produces the desired behavior [1].

1.2 DAJ

DAJ stands for Distributed algorithms in Java. DAJ was developed by Wolfgang Schreiner [7]. It is a toolkit for designing, implementing, testing, simulating, and visualizing distributed algorithms in Java. The toolkit consists of a Java class library with a simple programming interface that allows development of distributed algorithms based on a message passing model. [3].

An application developed with DAJ may run in one of three modes:

- Standalone, without visualization.
- Standalone, with visualization.
- As an applet embedded in a Web page [3].

The following are features of the programming model:

- Execution model consists of nodes that are linked by channels and each of them operates asynchronously and independently.
- Communication between the nodes which are linked by channels is through point-to-point messages.
- A node may send messages using channels and receive messages non-deterministically using channels [3].

The following are the features of the visualization environment of DAJ:

- The user can define the size of the visualization screen that displays the network. Nodes can be laid out in the network by specifying there location in the screen.
- States of nodes and channels are indicated by different colors of their boundaries and also by messages in a foot line.
- User can define what all information of internal states of nodes and channels to be displayed. This information is displayed by pop-up windows, when the mouse pointer hovers on a node or channel.
- The execution of the whole system can be slowed down, interrupted, performed step by step, and restarted [3].

The current state of each node and channel is shown by the color of their boundary during execution. For a node, green indicates that the node is ready for execution, red indicates that the node is blocked (waiting for a message on some channel) and blue indicates that the node has terminated execution. Similarly for a channel, gray indicates that the channel is empty, green indicates that the channel holds at least one message and red indicates that the channel is empty and the receiver node of the channel is waiting to receive a message [9].

Figure 1.1 shows the visualization of a distributed system having two processes. The two network nodes are linked by channels. A bulb on one side of the channel, where it touches the node indicates the receiver side. The nodes contain labels "1" and "2". The small numerical subscript of each label which is “7” for both the nodes denotes the current local time. Similarly the small number in the left upper corner of the window which is “10” denotes the global network time. The popup over the channel from process 1 to process 2 displays the state of the channel and its color green shows that it contains at least one message.

The visualization window has different buttons which can be used to control the execution of the system. Pressing the ‘*Run*’ button starts execution of the program. After the program execution has terminated, pressing the ‘*Reset*’ button will initialize the visualization again. Pressing the ‘*Walk*’ button slows down the pace of execution, whereas pressing the ‘*Interrupt*’ button suspends the execution. Pressing the ‘*Step*’ button allows the program execution to advance by one step. A step in the program execution means one communication operation. Pressing the ‘*Redraw*’ button redraws the screen, whereas pressing the ‘*Quit*’ button lets the visualization to terminate. However if the network is currently executing, then it has to be interrupted before pressing the ‘*Quit*’ button [9].

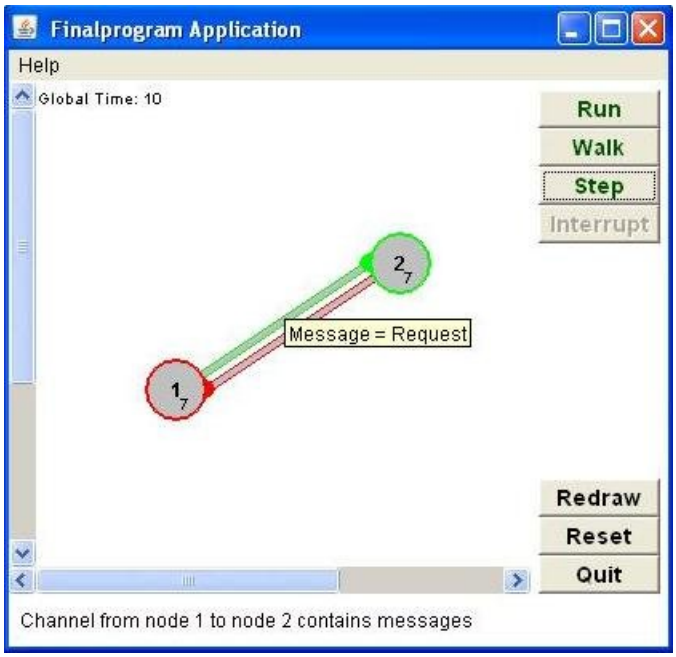

Figure 1.1 Simulation screenshot of DAJ

1.3 ANTLR

ANTLR stands for Another Tool for Language Recognition. It is a language tool that provides a framework for constructing recognizers, interpreters, compilers, and translators from grammatical descriptions containing actions in a variety of target languages [4]. It can also generate the source code for various language processing tools like lexers and parsers that can be used to analyze and transform input, in the language defined by the input grammar [4]. This feature of ANTLR saves a lot of time and resources as it automates significant portions of the effort involved in building language processing tools.

ANTLR's root construct is a grammar, which is essentially a list of rules that describe the structure of a particular language. From this list of rules, ANTLR generates a recursive-descent parser that recognizes sentences in that language [7]. The language might be a programming language or simply a data format, but ANTLR does not naturally know anything about the language described by the grammar. It is simply building a recognizer that can answers the question that, whether an input sequence a valid sentence according to the grammar? In other words, does the input follow the rules of the language described by the grammar? ANTLR grammars can optionally include *action code* written in what is termed the *target language* i.e. the implementation language of the source code generated by ANTLR [7]. These action codes must be embedded within the grammar to extract information, or to perform a translation, or deduce the incoming symbols in some other way according to the need [7].

In order to construct a translator using ANTLR, we have to first write a grammar that describes the overall syntactic structure of input language. The result of this effort is a recognizer which can answer yes or no as to whether the input is a valid expression or assignment. Then, we have to embed code among the grammar elements at appropriate positions to evaluate pieces of the expression and perform a translation [7]. For example, given an input, the translator must execute an action that converts the character to its integer value. For input 23+44, the translator must execute an action that adds the results from two previous action executions, namely, the actions that converted characters 23 and 44 to their integer equivalents. A grammar for example named '*Expression*' should be saved in a file also named the same i.e. '*Expression.g*' where '*.g*'

file extension indicates that it is a grammar file. For this grammar ANTLR will generate files as shown in table 1.1 [7].

Generated File	Description
ExpressionParser.java	The recursive-descent parser generated from the grammar Expr.g
Expression.tokens	The file consists of list of token names.
ExpressionLexer.java	The recursive-descent lexer generated the grammar Expr.g

Table 1.1 Files generated by ANTLR for a grammar '*Expression*'

1.4 ANTLRWorks

The easiest way to work with ANTLR grammars is to use ANTLRWorks, which provides a sophisticated development environment [7]. ANTLRWorks is a GUI development environment that sits on top of ANTLR and helps you edit, navigate, and debug grammars. This tool is very helpful for building grammars, resolving the errors, generating source code. The syntax diagram view of a rule makes it easy to understand exactly what the rule matches [5]. Perhaps most important, ANTLRWorks helps you resolve grammar analysis errors, which can be tricky to figure out manually. Some of the features of ANTLRWorks are as follows:

- It provides a grammar aware editor.
- It shows the syntax diagram view for a grammar.
- Helps in debugging a grammar
- Generates code for parser and lexer.
- It also provides decision lookahead visualization.
- Provides dynamic parse tree for a given input.

Figure 1.2, illustrates ANTLRWorks debugger. The debugger provides lot of information and always keeps the different views in sync. In this case, the grammar matches input identifier '*IncCnt*' with grammar element '*ID*'; the parse tree pane shows the implicit tree structure of the input. You can also use ANTLRWorks to generate code using the '*Generate*' menu's '*Generate Code*' option, which will generate code in the output directory of ANTLRWorks [7].

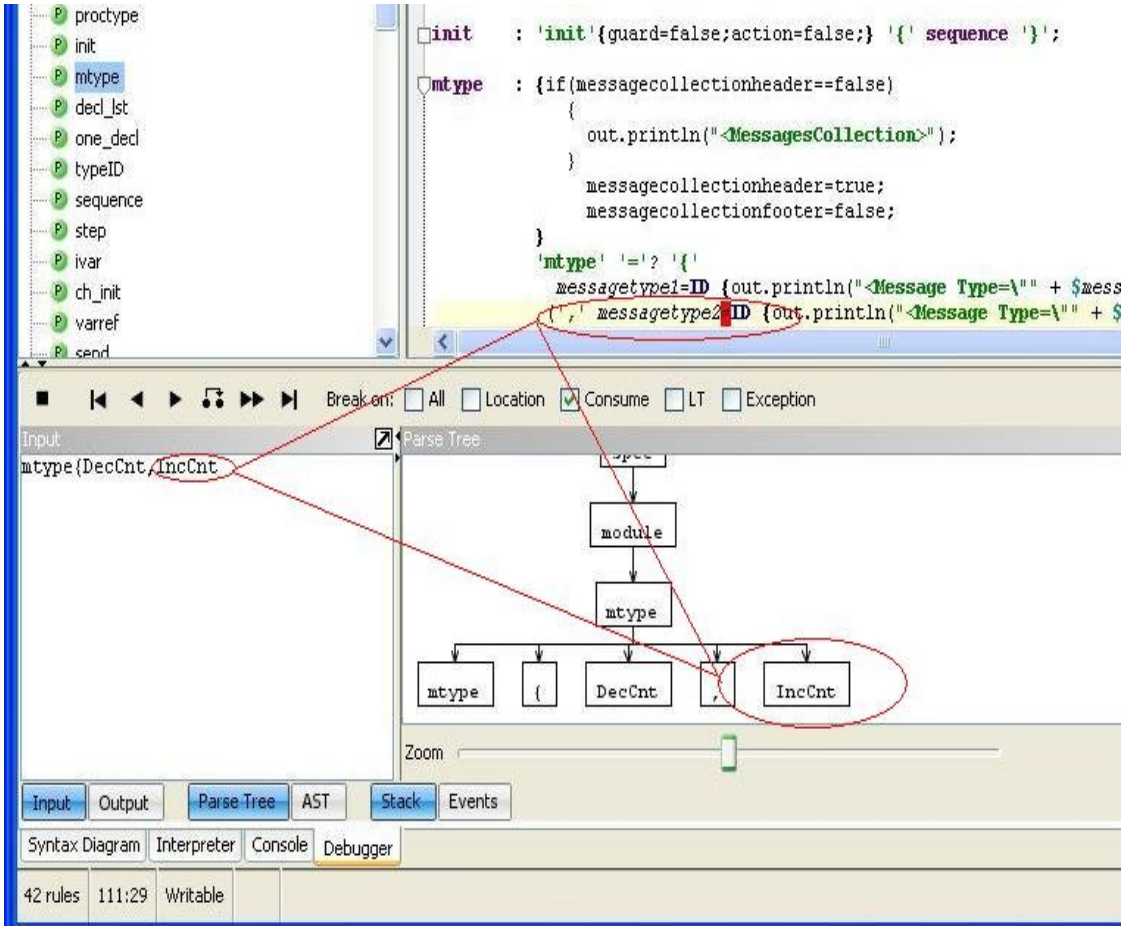

Figure 1.2 ANTLRWorks IDE in grammar debugging mode

1.5 StringTemplate

StringTemplate is a template engine which simply acts as a code generator that emits text using templates, which are really just "documents with holes" in them where you can stick values. StringTemplate is a java template engine for generating source code or any other formatted text output [6]. StringTemplate breaks up your template into chunks of text and attribute expressions, which are enclosed in dollar signs i.e. $*attribute-expression*$ or between less than and greater than symbols i.e. < *attribute-expression* >. StringTemplate ignores everything outside of attribute expressions, treating it as just text to spit out when you call [6]. For example, the following template has two chunks, a literal and a reference to attribute '*name*', which is a multivalued attribute having comma (',') as separator:

Hello <name; separator=", " >

This template can be used to print "Hello Bob, Ryan". The following is the example which does that.

```
StringTemplate template = new StringTemplate ("Hello, $name$");
template.setAttribute ("name", "Bob");
template.setAttribute ("name", "Ryan");
System.out.println (hello.toString ());
```

In order to use StringTemplate, we can directly create a template in code or we can load a template from a file or else we can load a single file with many templates which is called a template group file. There are many other features of String Template like templates can be declared recursive, templates can call other templates and pass arguments to them, attributes can be multi-valued etc.

Another distinctive StringTemplate language feature lacking in other engines is *lazy-evaluation* [11]. All the attributes of the template have to be defined in advance i.e. prior to the call, which writes the template to text. This is called "*push method*" whereas most template engines use the "*pull method*" [11]. But while defining the template attributes you can define the attributes in any order. For example if template ‘A’ has an attribute 'x' whose value is the output of another template ‘B’, we can either assign the value of template ‘B’ to attribute 'x' of template ‘A’ even before defining the attributes of ‘B’ or vice versa. In both ways, when template ‘A’ is asked to render itself to text, it will first evaluate ‘B’ and then its value is assigned to x, and then only A's value is calculated. So, lazy here means that the any order can be followed while defining the attributes of the template, but all the attributes (self as well as referenced attributes) have to be defined prior to the call which writes the template to text. Referencing attribute "a" does not actually invoke the data lookup mechanism until the template is asked to render itself to text [11].

1.6 Advantages

This application has two main advantages which are as follows:-

- The main advantage of this application is that by just writing a protocol in input model which is a variant of Promela, the simulation of the protocol can be visualized in DAJ, which has a nice visualization environment.
- The user need not know about the different classes of DAJ, in order to simulate and visualize a distributed algorithm using DAJ.

CHAPTER 2 - SYSTEM ANALYSIS

In this chapter, the system is analyzed by comparing the two different models, defining the input scope and structure, output obtained and also by specifying the requirements for the system. Initially, a comparison is done between the two different models used in the system i.e. Promela and DAJ. This comparison helped in understanding the similarities and differences in both models. It gave a clear understanding on the need of considering a subset of Promela as the input model. It also helped in what subset of Promela should be considered for translation. Later the structure and scope of input model is defined.

2.1 Comparison between Promela and DAJ

Both Promela and the Java class library of DAJ can be used to define distributed algorithms. The main similarity is that both uses a message passing model, i.e. the inter-processes communication in the models is through messages. The following are some of the similarities between the models, especially between the core constructs:-

- In Promela, processes are the distributed entities and can be defined using *proctype* keyword. Similarly, in DAJ, we can create processes using the class *Node* and the behavior of that process can be defined using the class *Program*.
- In Promela, a message can be defined using *mtype* keyword. In DAJ, messages are created using the class *Message*. These messages can have parameters.
- In Promela, a channel can be defined using *chan* keyword, and then the channel capacity has to be specified in order for it to hold more than one message at a time. The same channel can be used as both input as well as output channel by the same process. In DAJ, a channel can be created using the method '*link*' of the class '*Channel*'. This method takes two parameters which are the sender node and the receiver node and this channel can be used only as a uni-directional point-to-point channel. The channel can contain as many messages as required. There is no restriction on the maximum number of messages a channel can contain.

- The datatypes of variables used in Promela are int, bool, short, byte and we can also create arrays. The datatypes of variables in DAJ is basically that of Java as DAJ is implemented using Java.

Apart from these resemblances, both these models also differ in many aspects. The programming interface of DAJ is simple and very small when compared to that of Promela. It does not provide most of the features which Promela does. For example, DAJ does not provide process priorities, exclusive access to read and write to channels, timeout for the whole system, etc. Similarly there are few features which DAJ provides and Promela doesn't. For example Promela does not support process local time and global time for the system. Because of these differences, it is not possible for the whole of the Promela model to be considered for translation. Hence, there is a need to define an input scope and structure which can be easily translated to the output model. The following sections define the input structure, scope which is basically a variant of Promela. While designing the input scope, the primary focus has been to use most of the features possible of both the models and in order to do this; the input model defined may not follow some of the semantics of Promela model. Here onwards in the report, the input model would be referring to the model which is a variant of Promela.

2.2 Input Structure and Scope

The input structure and scope of this application confines to a variant of the subset of Promela model defined as per the following constraints. The basic structure of the input model consists of four sections. A model will have a message declaration section where in all the messages used are defined. Then, there will be a channel declaration section where in all the channels used and how they link processes is defined. Then, there can be a global variable declaration section where in all the global variables used are defined. The final section is the process declaration section where in all the processes are defined. For each process, there are two sections among which the first is a local variable declaration section where all the local variables used in the process is defined. The second section within a process is the guard action declaration sections where in the actual functionality of the process is defined, which are the guards and actions.

<<Messages Declaration>>

<<Channels Declaration>>

<<Global Variables Declaration>>

<<Processes Declaration

<<Local Variables Declaration>>

<<Guard Actions Declaration>>

>>

Following sections discuss the four sections of the input model along with their usage and constraints.

2.2.1 Messages

The first section should be the declaration of messages. A message can be declared by using the '*mtype*' keyword followed by curly brackets containing the message name followed by open and close parentheses, which can contain the different parameters which the message can have. Each parameter if it exists should have an integer data type only and followed by a parameter name. If more than one parameter have to be declared then they have to be separated by a ','. If more than one message has to be declared, the message names have to be separated by a ','. Example:-

mtype {Token(),Request(int value, int flag)}

In Promela in order to send more than one value as parameter for a single send operation, a channel has to be defined so as to carry more than one parameter. For example, '*chan q = {int,int};*'. Here channel q can carry two parameters for a single send operation. For example '*q!2,3*'. On the other hand, a message in DAJ is an object of class '*Message*' and can have as many parameters as required. So in order to send more than parameter for a single '*send*' operation, a message has to be defined such that it can have more than one parameter.

2.2.2 Channels

The channels section should follow immediately after the messages section. A channel can be only uni-directional and is between two particular processes and can be used to send or receive messsges which are already defined. A channel can be defined by using the '*chan*' keyword and has syntax as follows:-

chan channelname sourceprocessname destinationprocessname;

Example:-chan channel12 node1 node2;

The sourceproccessname and destinationprocessname should match with the processes defined. In the above example channel12 is the channel using which node1 can only send messages to node2 only. All the channels should be defined in the similar fashion. This syntax of channel declaration is different from that of Promela, because in DAJ each channel is between only two specified processes, which form the source and destination for that channel.

2.2.3 Global Variables

This section should be followed immediately after channels section. Here all the global variables needed for the application have to be defined. The datatypes allowed are int, short, byte and boolean. The variables must be initialized with some initial value. Integer arrays can also be defined.

Example:-

```
int flag = 0, start = 1, hastoken = 0;
boolean first = false, iswriter = true;
int hastoken[2] = 0;
```

2.2.4 Processes

This is last section of the input model where all the processes participating should be defined. A process can be defined using the '*proctype*' keyword as follows:-

```
proctype  proccessname()
{
   Local Variables Section
   Guard Action Section
}
```

A process cannot have parameters. The process body consists of two sections local variables section and guard action section.

2.2.4.1 Local Variables

This section should be the first section in the process body. Here all the local variables needed for the process have to be defined. The datatypes allowed are int, short,

byte and boolean. The variables must be initialized with some initial value. Integer arrays can also be defined.

Example:-

```
int flag = 0, start = 1, hastoken = 0;
boolean first = false, iswriter = true;
int hastoken[2] = 0;
```

2.2.4.2 Guard Actions

This section is followed after the local variables section. This consists of a single do loop which can have many guards and there corresponding actions. The basic outline structure of this section is as follows.

```
do
  :: atomic{ guard -> actions }
  :: atomic{ guard -> actions }
  .
  .
od
```

The guard section consists of either one or more logical statements or a message receive statement. The receive statement is the same as that in Promela i.e. channel name followed by question mark (?) followed by the message name. The actions section consists of one or more action statements each ending with a semicolon (';'). An action statement can be a send statement or an assignment statement or an increment or decrement statement or an '*if*' statement only. In the send statement all the parameter values of the message should be defined i.e. if a message has two parameters, then both the parameter values have to be defined. In order to refer to the parameter value and use it in other actions, the message name to which the parameter belongs has to be followed by 'obj' followed by '.' and finally followed by the parameter name defined in message declaration. The following example illustrates this.

Example:-

```
mtype {IncCnt(int value1)}
channel12!IncCnt(2)
flag = IncCntobj.value1;
```

'*If*' loop can also be an action. The syntax of '*if*' loop is same as that of Promela, but the semantics is different. In Promela in the '*if*' loop, if none of the guards are true, the process blocks until at least one of the guard is true. But in DAJ a process will only get blocked when it tries to receive message on an empty channel. Hence, the semantics of if loop in the input model have been considered to be that of if loop semantics in java. The first guard of the '*if*' loop forms the '*if*' part and rest of the guards form the '*else if*' part. The following is an example of guard action section of the input model.

Example:-

```
do
::atomic{hastoken==1 - >channel23!Token();hastoken=  0;reqreceived=0;}
::atomic{hastoken == 1 && incs==0 && reqreceived == 0 && i < 1- incs = 1;
i++; }
::atomic{ channel12?Token - > hastoken = 1;reqsent =0; }
::atomic{ hastoken == 0 && reqsent == 0 - > channel21!Request(); reqsent
=1;}
::atomic{ flag == 0 -> channel12!IncCnt(2);flag =1;}
 ::atomic{ channel21?DecCnt -> if
                                        :: DecCntobj.value1 == 0 -> flag = 0;
                                    fi;}
::atomic{ channel32?Request - > reqreceived = 1;}
::atomic{ incs == 1 -> incs = 0; }
od
```

2.2.5 Keywords

Table 2.1 shows all the keywords which can be used in the input model.

proctype	mtype	chan	receive	send	int	boolean
True	false	atomic	Do	od	if	fi
Short	byte					

Table 2.1 Input model keywords

2.2.6 Operators

Table 2.2 shows all the different operators which can be used in the input model.

+	-	++	--	<=
*	/	>	<	>=
==	!=	&&	||	

Table 2.2 Input model operators

Figure 2.1 shows a sample valid input program for this application with different sections highlighted.

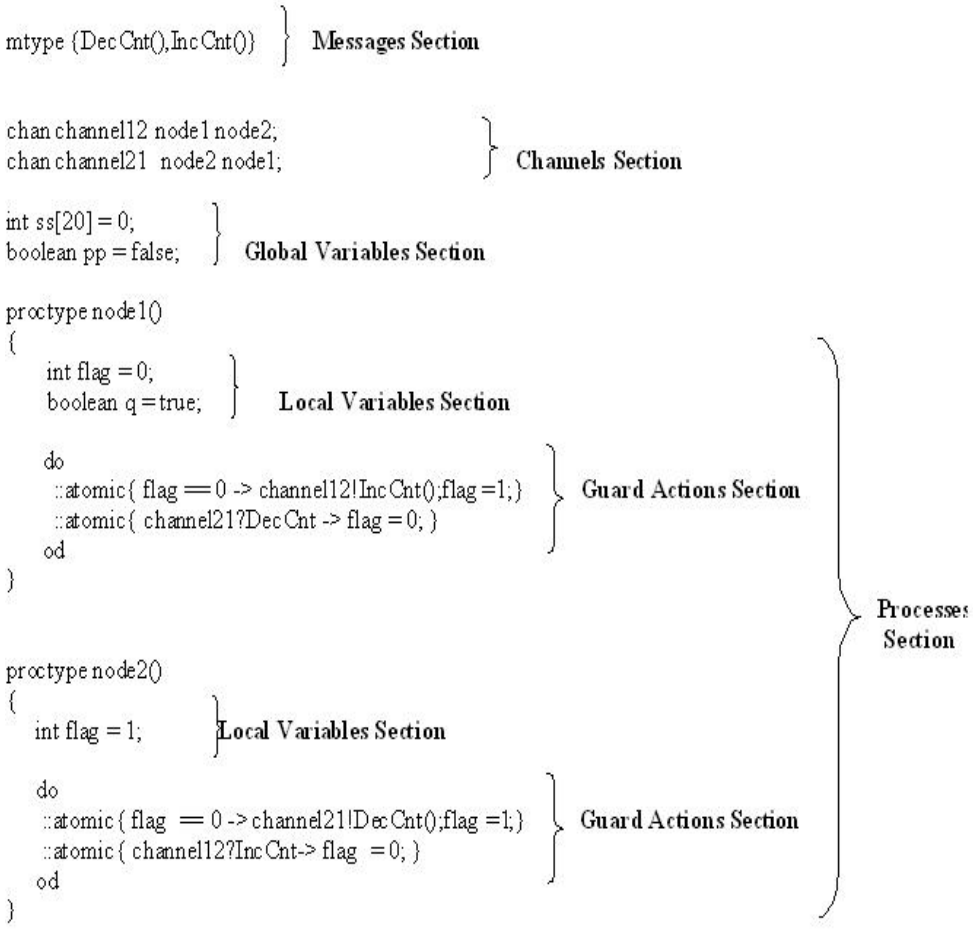

Figure 2.1 Input Model Structure

Appendix A contains the complete ANTLR grammar for the input model.

2.3 Software Specifications

This application has been developed using different software's whose technical specification details are as follows:-

- ANTLR 3.1.3
- StringTemplate 3.0
- Java SE 6
- DAJ 1.0.2

CHAPTER 3 - IMPLEMENTATION

3.1 Design

The translation of input model to output model is done in three stages. In the first stage the input is parsed and translated into an intermediate format which is xml. In order to parse the input, an ANTLR grammar has been written. To this grammar, java code has been embedded for it to translate the input in to an xml output. This grammar is saved in a file called '*promela.g'*. Using ANTLRWorks tool and the written grammar file, parser and lexer code are generated which are '*promelaParser.java'* and '*promelaLexer.java'*. Using these two generated files, the input can be translated into an xml format and saved in a file called '*intermediateXMLFile.xml'*. The obtained xml file consists basically of the entire input model put in an xml format and some more information, which would help in translating the input model to output model. An XML Document Type Definition (DTD) has been defined for the structure of the intermediate xml file. This is saved in a file called *'intermediateXMLFileDTD.dtd'*. The document type definition helps in easily traversing the obtained xml file which is '*intermediateXMLFile.xml'* and get the required information.

In the second stage the intermediate xml format is translated to output model. In order to do this, a String Template file is defined which consists of the code templates of the output model. Then traversing through the xml file, the code templates are filled with the information present in the xml file and the output model is generated, which is saved in a file called *'FinalProgram.java'*. The xml file is traversed more than once during this process.

In the third stage, the obtained output model is simulated over DAJ. In order to do this, first the *'FinalProgram.java'* is compiled and then a jar file named *'FinalProgram.jar'* created which is composed of *'FinalProgram.class'* and all other class files of DAJ. The jar file when executed simulates the output model in the visualization environment of DAJ, where the user can simulate the model in step by step motion, slowed down motion or usual pace. Figure 3.1 shows the whole workflow of the tool. It depicts all the files used. The following sections of this chapter describe the different files used and generated in the system work flow in detail.

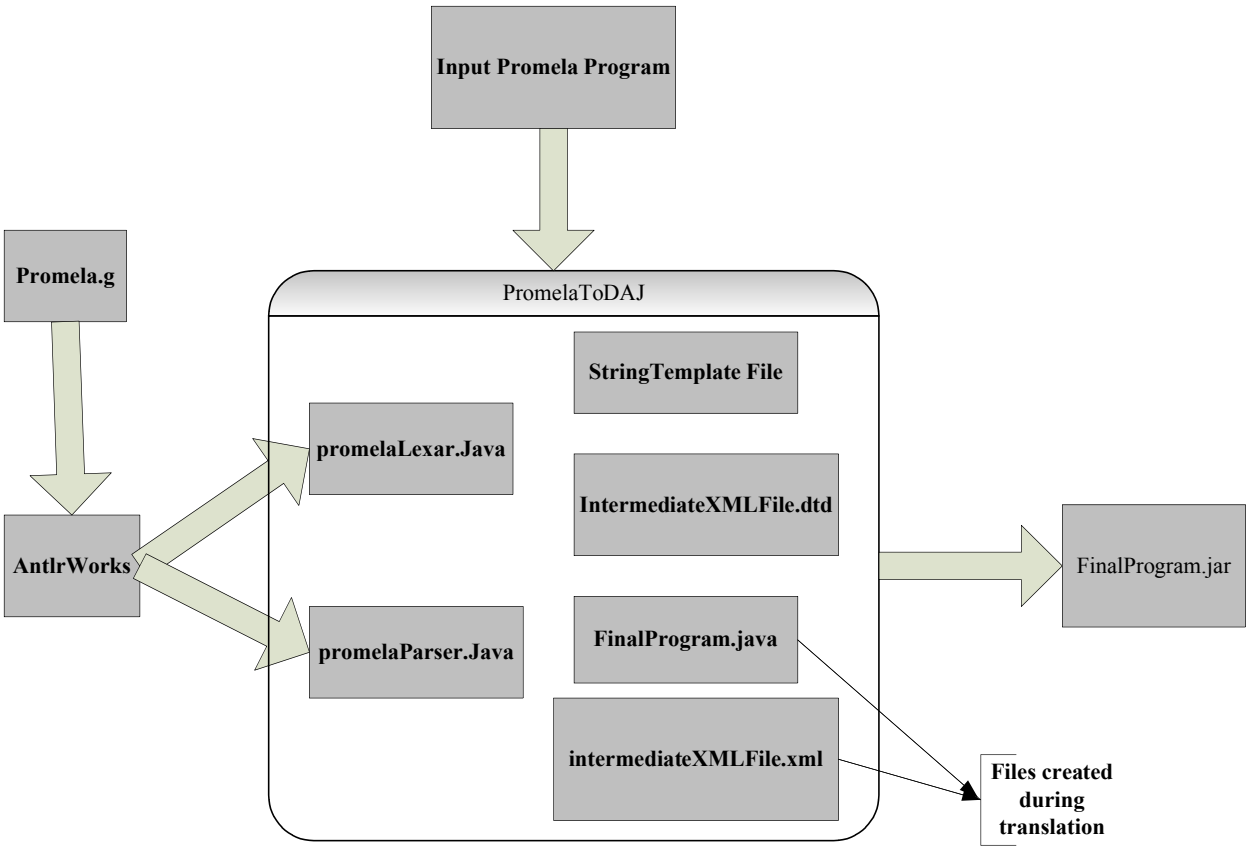

Figure 3.1 System Work Flow

3.2 User Interface

The user interface of this application is very simple and allows the user to give a file as input, which consists of a distributed algorithm defined in the input model and creates a jar file named '*FinalProgram.jar*' in current directory. Figure 3.2 shows the screenshot of the user interface. When the user clicks the '*Open a Promela Model File*' button a file open file dialog pops up and the user can select the input file and click open. Then the text box populates with the path of selected file. When the user clicks the '*Create Jar*' button the 'FinalProgram.jar' is created. The user can then run the obtained jar file which simulates the output model in the DAJ visualization environment.

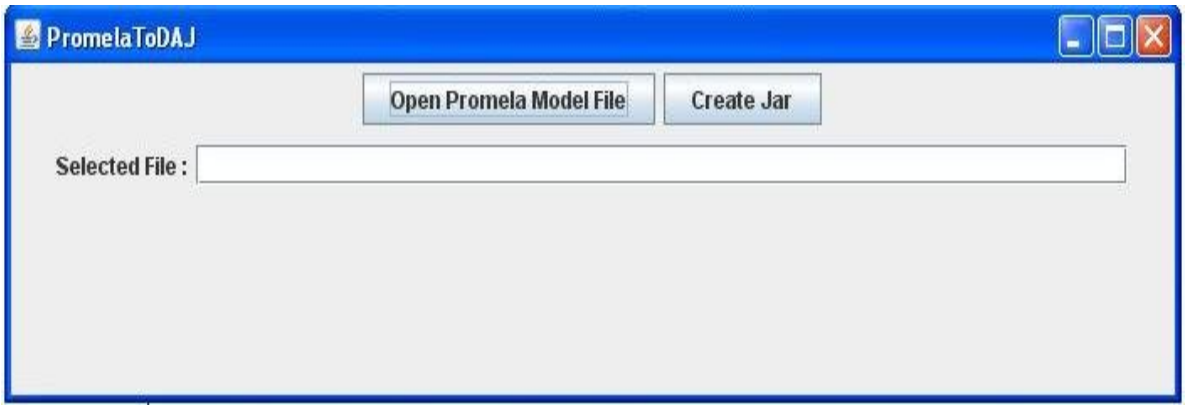

Figure 3.2 User Interface Screenshot

3.3 Promela.g

This is the ANTLR grammar file which recognizes the input and translates the input into an xml format. This file consists of ANTLR grammar rules as well the action code written in Java embedded at appropriate locations. This file is written using the AntlrWorks tool and then source code is generated using the generate code option of AntlrWorks. This generates two files, '*promelaLexer.java*' and '*promelaParser.java'* in the output folder of AntlrWorks. The embedded actions would basically create a file called '*intermediateXMLFile.xml*' in the current directory and then write the output to that file. The embedded action code generates the xml output whose format is as defined by the *'intermediateXMLFileDTD.dtd'*. Appendix A contains the ANTLR grammar for the input model without the embedded actions. The generated parser and lexer files from the grammar are used along with two classes of ANTLR, which are ANTLRFileStream and CommonTokenStream. The ANTLRFileStream class is used to take an input stream to be parsed. CommonTokenStream class is used to collect the token given by the lexer. The following code snippet shows how this is done.

```
ANTLRFileStream fileinput = new ANTLRFileStream(input filename);
promelaLexer lexer = new promelaLexer(fileinput);
// Create a stream of tokens fed by the lexer
CommonTokenStream tokens = new CommonTokenStream(lexer);
// Create a parser that feeds off the token stream
promelaParser parser = new promelaParser(tokens);
// Begin parsing at start rule spec
```

parser.spec();

3.4 IntermediateXMLFileDTD.dtd

This file defines the structure of the generated xml file. It is used in order to parse and traverse the generated xml file. The structure closely resembles the input model structure defined in chapter 2. *Application* is the root xml element which consists of *ProcessesCollection* element, *ChannelsCollection* element, *MessagesCollection* element and *GlobalVariablesCollection* element.

The following code snippet shows the complete document type definition of the generated xml file. It describes what elements it can have and what all attributes each element has. For example, a *ProcessesCollection* can have more than one *Process* elements. Each Process element can have a *LocalVariablesCollection* element followed by one *GuardActionCollection* element, and apart from these elements it can also have two attributes which are *Name* and *ProcessID*.

```
<!ELEMENT Application (ProcessesCollection, ChannelsCollection,
MessagesCollection, GlobalVariablesCollection)>
<!ELEMENT ProcessesCollection (Process+)>
<!ELEMENT Process (LocalVariablesCollection, GuardActionCollection)>
<!ATTLIST Process Name CDATA #REQUIRED>
<!ATTLIST Process ProcessID CDATA #REQUIRED>
<!ELEMENT LocalVariablesCollection (LocalVariable*)>
<!ELEMENT LocalVariable (EMPTY)>
<!ATTLIST LocalVariable Name CDATA #REQUIRED>
<!ATTLIST LocalVariable Value CDATA #REQUIRED>
<!ATTLIST LocalVariable DataType CDATA #REQUIRED>
<!ELEMENT GuardActionCollection (GuardAction*)>
<!ELEMENT GuardAction (Guards*, Actions*)>
<!ELEMENT Guards (Guard*)>
<!ELEMENT Guard (#PCDATA)>
```

<!ATTLIST Guard Type CDATA #IMPLIED>

<!ELEMENT Actions (Action, IfAction*)>*

<!ELEMENT Action (#PCDATA)>

<!ATTLIST Action Type CDATA #IMPLIED>

<!ELEMENT IfAction (Guards, Actions*)>*

<!ELEMENT ChannelsCollection (Channel)>*

<!ELEMENT Channel (EMPTY)>

<!ATTLIST Channel Name CDATA #REQUIRED>

<!ATTLIST Channel SourceProcess CDATA #REQUIRED>

<!ATTLIST Channel DestinationProcess CDATA #REQUIRED>

<!ELEMENT MessagesCollection (Message+)>

<!ELEMENT Message (MessageParameter)>*

<!ATTLIST Message Type CDATA #REQUIRED>

<!ELEMENT MessageParameter (EMPTY)>

<!ATTLIST MessageParameter DataType CDATA #REQUIRED>

<!ATTLIST MessageParameter Name CDATA #REQUIRED>

<!ELEMENT GlobalVariablesCollection (GlobalVariable)>*

<!ELEMENT GlobalVariable (EMPTY)>

<!ATTLIST GlobalVariable Name CDATA #REQUIRED>

<!ATTLIST GlobalVariable Value CDATA #REQUIRED>

<!ATTLIST GlobalVariable DataType CDATA #REQUIRED>

3.5 IntermediateXMLFile.xml

This file is generated during the second stage of the translation process. This file consists of all the information of the input model along with some other information which would be helpful for generating output model, put in an xml format whose structure is defined by the '*IntermediateXMLFileDTD.dtd*' file. Following is an example of the xml file generated for an input model having two processes.

Example:-

<?xml version="1.0"?>

<!DOCTYPE IntermediateXmlFileDTD SYSTEM "IntermediateXmlFileDTD.dtd" >

<Application>
<MessagesCollection>
<Message Type="DecCnt">
<MessageParameter DataType="int" Name="value"></MessageParameter>
</Message>
<Message Type="IncCnt">
<MessageParameter DataType="int" Name="value"></MessageParameter>
</Message>
</MessagesCollection>
<ChannelsCollection>
<Channel Name="channel12" SourceProcess="node1" DestinationProcess="node2">
</Channel>
<Channel Name="channel21" SourceProcess="node2" DestinationProcess="node1">
</Channel>
</ChannelsCollection>
<ProcessesCollection>
<Process Name="node1" ProcessID="1">
<LocalVariablesCollection>
<LocalVariable Name="flag" Value="0" DataType="int"></LocalVariable>
</LocalVariablesCollection>
<GuardActionCollection>
<GuardAction>
<Guards>
<Guard>flag==0</Guard>
</Guards>
<Actions>
<Action Type="send">channel12!IncCnt(5)</Action>
<Action>flag=1</Action>
</Actions>
</GuardAction>
<GuardAction>

<Guards>
<Guard Type="recieve">channel21?DecCnt</Guard>
</Guards>
<Actions>
<Action>flag=DecCntobj.value</Action>
</Actions>
</GuardAction>
</GuardActionCollection>
</Process>
<Process Name="node2" ProcessID="2">
<LocalVariablesCollection>
<LocalVariable Name="flag" Value="0" DataType="int"></LocalVariable>
</LocalVariablesCollection>
<GuardActionCollection>
<GuardAction>
<Guards>
<Guard>flag==5</Guard>
</Guards>
<Actions>
<Action Type="send">channel21!DecCnt(0)</Action>
<Action>flag=0</Action>
</Actions>
</GuardAction>
<GuardAction>
<Guards>
<Guard Type="receive">channel12?IncCnt</Guard>
</Guards>
<Actions>
<Action>flag=IncCntobj.value</Action>
</Actions>
</GuardAction>

</GuardActionCollection>

</Process>

</ProcessesCollection>

</Application>

3.6 String Template File

A String Template file is defined which consists of many templates for different possible constructs of the output model. These templates are filled in using the information obtained while traversing the intermediate xml file. Following is an example of the template for declaring global variables in the output model. This template takes in parameters, and using these values, String Template engine fills the holes in the templates, when the template is printed.

Example:-

globalvariabledeclaration(variabledatatype,variablename,variablevalue,isarray,arraysize,arrayvalue) ::= <<

<if(arraysize)>

<variabledatatype> [] <variablename> = new <variabledatatype>[<arraysize>];

<else>

<variabledatatype> <variablename> = <variablevalue>;

<endif>

>>

The above template takes 6 parameters. It can be used for declaring normal variables as well as arrays. The '*if*' part of the template is for declaring arrays and the '*else*' part of the template is for declaring variables other than arrays. If '*isarray*' parameter is false then the '*else*' part of the template is used or else the '*if*' part of the template is used. If the values given to this template for the corresponding parameters are int, flag, 0, false, 0, and 0, then the output of the template would be '*int flag = 0;*'. In the similar fashion many templates have been written for the different possible constructs of the output model.

CHAPTER 4 - TESTING

The application has been tested by giving different input programs covering most of the input scope. Input test programs were written such that, they have different topologies, different processes, having different number of processes, channels, messages and which cover most of the input scope. For these input programs output models are generated and checked whether the desired simulation of the input model is obtained or not. This chapter discusses two different test programs and the output obtained for each test program. The code for the input model of each program is defined in this chapter. The intermediate generated xml file and the obtained output model for both the programs is given in Appendix B.

4.1 Test Program 1 – Token Ring Protocol

In this protocol 4 processes participate in a ring topology and try to get in to critical section. In order to go to critical section they require a token which is initially present with one among the 4 processes. If a process wants a token to go critical section, it requests the process to the left by sending a '*Request*' message. Upon receiving a '*Request*' message from right, a process sets a flag that it has received a request for the token and if it has token, it sends the 'Token' message. If it has no token then it checks whether it has sent any request messages to its right before and if so they have been answered back with a token. If it has sent any previous request message then it would not send a new request message. Any process having the token is constrained to go to critical section only once. In the ring topology '*Token*' messages move in anti clockwise direction and '*Request'* messages move in clockwise direction. For this kind of protocol an input model is written which is shown below:-

```
mtype {Token(),Request()}
chan channel12 node1 node2;
chan channel14 node1 node4;
chan channel21 node2 node1;
chan channel23 node2 node3;
chan channel34 node3 node4;
```

```
chan channel32 node3 node2;
chan channel41 node4 node1;
chan channel43 node4 node3;
proctype node1()
{
  int hastoken = 0,incs = 0, reqsent = 0, reqreceived = 0, i = 0;
   do
      ::atomic{hastoken==1&&incs==0&&reqreceived==1->
channel12!Token();hastoken =  0;reqreceived=0; i =0;}
       ::atomic{ hastoken == 1 && incs==0 && reqreceived == 0 && i < 1 ->
incs = 1; i++; }
      ::atomic{ channel41?Token -> hastoken = 1; reqsent =0;}
      ::atomic{ hastoken == 0 && reqsent == 0 -> channel14!Request(); reqsent
=1;}
      ::atomic{ channel21?Request -> reqreceived = 1;}
      ::atomic{ incs == 1 -> incs = 0; }
    od
}
 proctype node2()
{
  int hastoken = 1,incs = 0,reqsent = 0, reqreceived = 0, i =0  ;
  do
     ::atomic{ hastoken == 1 && incs==0 && reqreceived == 1 ->
channel23!Token();hastoken = 0;reqreceived=0;i =0;}
    ::atomic{ hastoken == 1 && incs==0 && reqreceived == 0 && i < 1-> incs
= 1; i++; }
    ::atomic{ channel12?Token -> hastoken = 1;reqsent =0; }
    ::atomic{ hastoken == 0 && reqsent == 0 -> channel21!Request(); reqsent
=1;}
    ::atomic{ channel32?Request -> reqreceived = 1;}
    ::atomic{ incs == 1 -> incs = 0; }
```

```
 od
}
proctype node3()
{
 int hastoken = 0,incs = 0,reqsent = 0, reqreceived = 0, i =0  ;
 do
   ::atomic{ hastoken == 1 && incs==0 && reqreceived == 1 -> channel34!Token();hastoken = 0;reqreceived=0;i =0;}
   ::atomic{ hastoken == 1 && incs==0 && reqreceived == 0 && i < 1-> incs = 1; i++; }
   ::atomic{ channel23?Token -> hastoken = 1;reqsent =0; }
   ::atomic{ hastoken == 0 && reqsent == 0 -> channel32!Request(); reqsent =1;}
  ::atomic{ channel43?Request -> reqreceived = 1;}
  ::atomic{ incs == 1 -> incs = 0; }
 od
}
proctype node4()
{
 int hastoken = 0,incs = 0,reqsent = 0, reqreceived = 0, i =0  ;
 do
    ::atomic{ hastoken == 1 && incs==0 && reqreceived == 1 -> channel41!Token();hastoken = 0;reqreceived=0;i =0;}
    ::atomic{ hastoken == 1 && incs==0 && reqreceived == 0 && i < 1-> incs = 1; i++; }
    ::atomic{ channel34?Token -> hastoken = 1;reqsent =0; }
    ::atomic{ hastoken == 0 && reqsent == 0 -> channel43!Request(); reqsent =1;}
    ::atomic{ channel14?Request -> reqreceived = 1;}
    ::atomic{ incs == 1 -> incs = 0; }
 od
```

}

For this input model, the output model is generated, which is then simulated and the desired simulation of the input model is obtained. Figure 4.1 shows a screenshot of the simulation of output model.

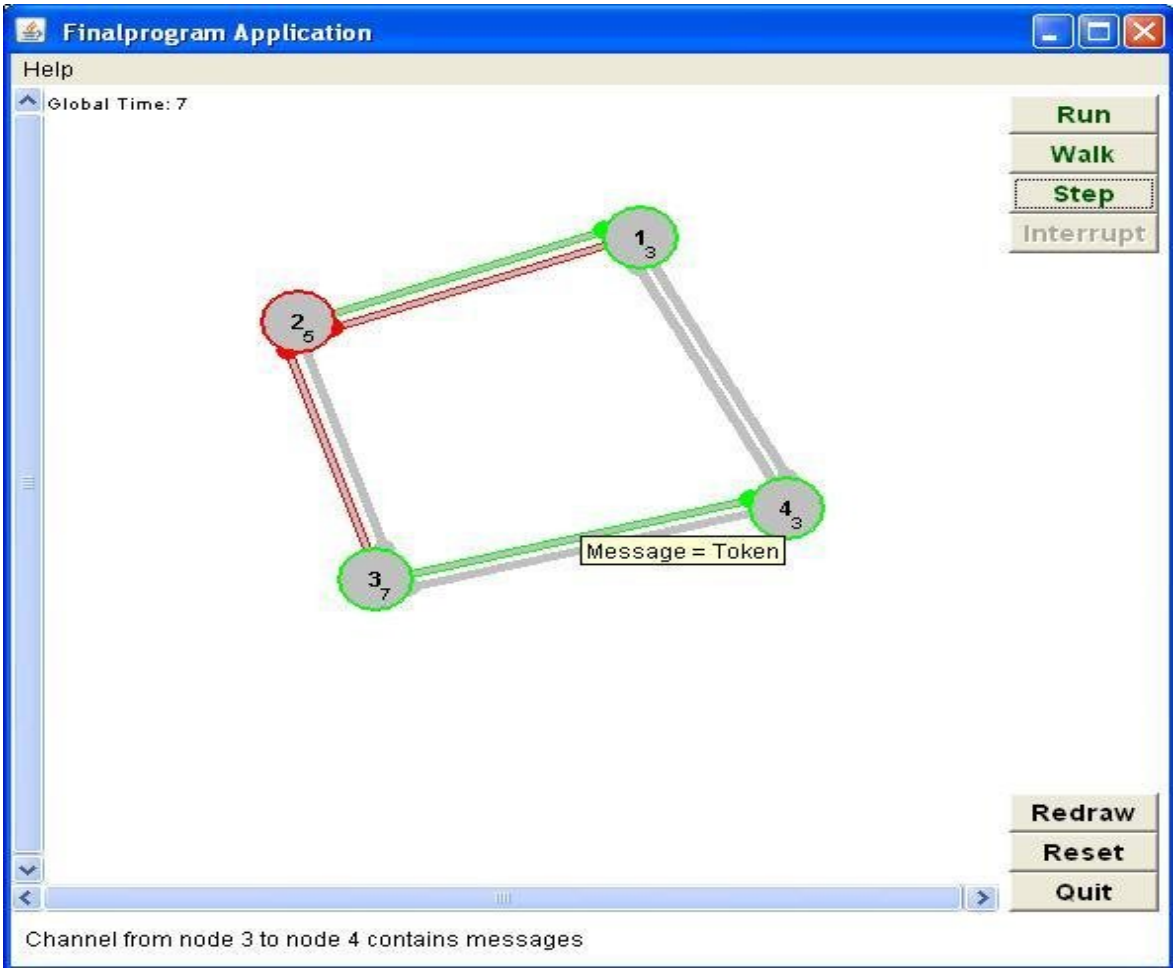

Figure 4.1 Simulation screenshot of Four Process Token Ring Protocol

4.2 Test Program 2 – Six Process in a Tree Topology

This is a simple protocol where in six processes form a tree, with process1 having process 2 and process 3 as its children, process 2 having process 4 as its child and process 3 having process 5 and process 6 as its children. It's a simple protocol where in the root process sends a message '*M*' to all its children and waits for an acknowledgement from all its children. Only

when it receives acknowledgements from all its children a process acknowledges back to its parent. In this case process 3 replies will acknowledge back to process 1 only when it receives a acknowledgement from its child processes i.e. process 5 and process 6. The following is the input model written for this protocol.

```
mtype {M(),ack(),ackr(),ackv()}
chan channel12 node1 node2;
chan channel21 node2 node1;
chan channel13 node1 node3;
chan channel31 node3 node1;
chan channel24 node2 node4;
chan channel42 node4 node2;
chan channel35 node3 node5;
chan channel53 node5 node3;
chan channel63 node6 node3;
chan channel36 node3 node6;
proctype node1()
{
  int flag = 1,ackrecv1 =0, ackrecv2 = 0;
    do
     ::atomic{ flag == 1 -> channel12!M();channel13!M();flag = 0;}
    ::atomic{ channel21?ackr -> ackrecv1 = 1;}
   ::atomic{ channel31?ackv -> ackrecv2 = 1;}
  ::atomic{ ackrecv1 ==1 && ackrecv2 == 1 -> flag = 1; ackrecv2 = 0 ; ackrecv1 = 0;}
  od
}
proctype node2()
{
  int flag = 0;
  do
   ::atomic{ flag  == 1 -> channel21!ackr();flag = 0;}
```

```
    ::atomic{ channel12?M-> flag  = 2; }
    ::atomic{ flag == 2 -> channel24!M();flag = 3;}
    ::atomic{ channel42?ack -> flag = 1;}
   od
  }
  proctype node3()
  {
    int flag = 0, ackrecv1 = 0, ackrecv2 =0;
    do
    ::atomic{ ackrecv1 ==1 && ackrecv2 == 1 -> flag = 1; ackrecv1 = 0 ; ackrecv2 = 0;}
    ::atomic{ flag  == 1 -> channel31!ackv();flag =0;}
   ::atomic{ channel53?ackr -> ackrecv1 = 1;}
  ::atomic{ channel63?ackv -> ackrecv2 = 1;}
   ::atomic{ channel13?M -> flag = 2;}
   ::atomic{ flag == 2 -> channel35!M(); channel36!M(); flag = 3;}
  od
}
  proctype node4()
  {
    int flag = 0;
     do
      ::atomic{ flag == 1 -> channel42!ack();flag = 0;}
     ::atomic{ channel24?M -> flag = 1}
    od
  }
  proctype node5()
  {
    int flag = 0;
     do
      ::atomic{ flag == 1 -> channel53!ackr();flag = 0;}
       ::atomic{ channel35?M -> flag = 1}
```

```
    od
  }
proctype node6()
  {
   int flag = 0;
    do
      ::atomic{ flag == 1 -> channel63!ackv();flag = 0;}
      ::atomic{ channel36?M -> flag = 1}
    od
  }
```

For this input model, the output model is generated, which is then simulated and the desired simulation of the input model is obtained. Figure 4.2 shows a screenshot of the simulation of output model.

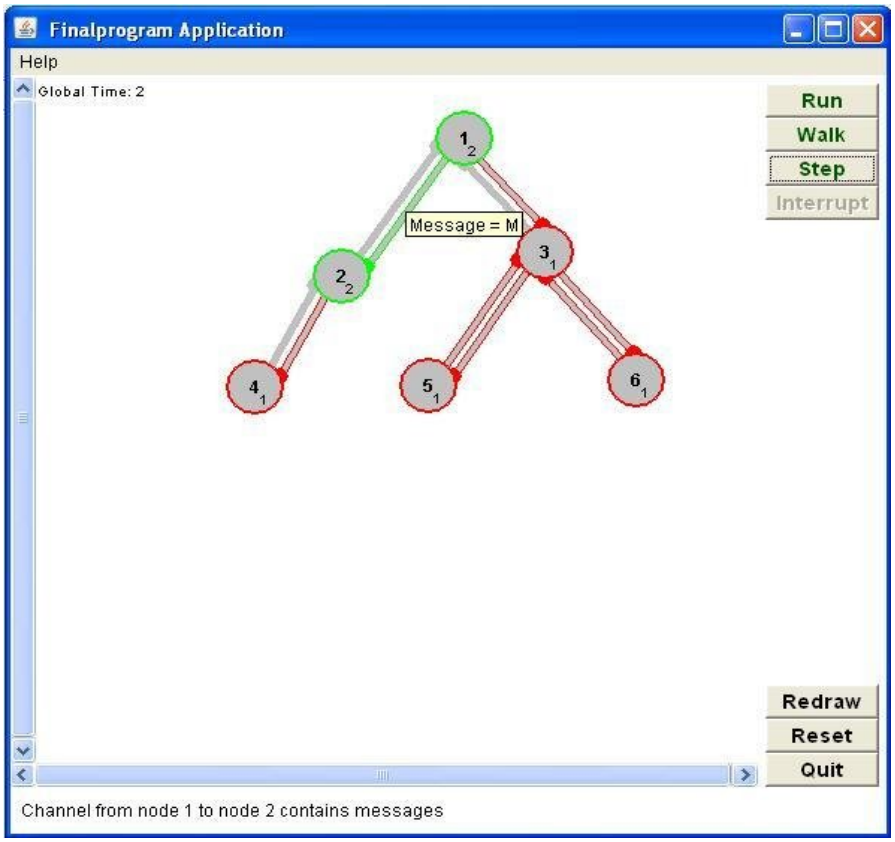

Figure 4.2 Simulation Screenshot of Six Processes in a Tree Topology Protocol

CHAPTER 5 - RESULTS AND CONCLUSION

This Project was developed with an intention to translate an input model which is a variant of Promela model to that of a model written using the Java class library of DAJ and moreover simulate the obtained model using DAJ. This goal has been achieved and a user interface has developed using which, user can give an input model as input and obtain the output model. The application has also been tested with many test programs having different topologies, different processes, different number of processes, channels, messages and which cover most of the input scope. The following sections of this chapter discuss the limitations of this application and also suggest ways in which the present application can be enhanced for more functionalities and also define the scope for future work.

5.1 Limitations

- The programming interface of DAJ is small when compared to Promela and this limits the important features provided by Promela.
- The major limitation of this application is on the input domain, which is very small when compared to that of Promela.
- The application cannot verify the correctness of the input model.
- Asserting system global conditions cannot be done, which is very important to check for the correctness of the protocol.
- The User interface is very simple and it does not provide any support for development of input models like displaying syntax errors of the input model etc.

5.2 Scope for Future Work

- Most of the limitations can be removed if the programming interface of DAJ can be expanded to match the interface provided by Promela. This paves the path for future work which can be done to make this application a better tool.
- Apart from this, a more sophisticated user interface can also be designed which would help in displaying the syntax errors of the input model, verification of the input model and help in development of input model.

- The input domain can be expanded to include more features of DAJ which are not present in Promela. For example, accessing local time of the processes and global time of the system.

REFERENCES

1. Promela Manual.
   http://spinroot.com/spin/Manual.html
2. Promela
   http://en.wikipedia.org/wiki/Promela
3. DAJ
   http://www.risc.uni-linz.ac.at/software/daj/report/index_1.html
4. ANTLR
   http://www.antlr.org/
5. ANTLRWorks
   http://www.antlr.org/works/index.html
6. StringTemplate - Documentation
   http://www.antlr.org/wiki/display/ST/Introduction
7. Terence Parr (2007). *The Definitive ANTLR Reference*. The Pragmatic Bookshelf
8. Wolfgang Schreiner – developer of DAJ
   http://www.risc.uni-linz.ac.at/people/schreine/
9. DAJ – Executing a program.
   http://www.risc.uni-linz.ac.at/software/daj/report/index_3.html
10. ANTLR Tutorial
    http://www.antlr.org/wiki/display/CS652/Meet+my+little+friends,+ANTLR+and+StringTemplate
11. StringTemplate - Introduction
    http://www.stringtemplate.org/about.html

Appendix A - ANTLR grammar for Input Model

Here is the complete ANTLR grammar for the input model without the actions embedded. An input program is a valid if it confines to this grammar.

```
grammar promela;
options {backtrack=true;memoize=true;language=Java;}
WS  :   (' '|'\t'|'\n'|'\r')+ {skip();} ;
ID      : ('a'..'z' |'A'..'Z' |'_' ) ('a'..'z' |'A'..'Z' |'_' |'0'..'9' )* ;
NUMBER      : '0'..'9';
constant : 'true' | 'false' | 'skip' | NUMBER(NUMBER) *;
spec    : module+ EOF;
module :      proctype
            | init
            | mtype
            | decl_lst ;
separator  :    ';'   |  '->';
proctype:  'proctype' ID '(' ')'
            '{' sequence * '}';
mtype  : 'mtype' '='? '{'ID'(' ( typeID ID)?( ',' typeID ID)*  ')'
                        ( ',' ID '('( typeID ID)?(','typeID ID )*')')*'}'  separator?   ;
decl_lst : one_decl( separator  one_decl ) * separator? ;
one_decl : typeID ivar (',' ivar )*;
typeID:  'boolean' | 'int' | 'mtype' | 'chan' | 'short' | 'byte' ;
sequence: step( separator    step )* separator? ;
step :  one_decl | stmnt;
ivar :   ID ( '[' constant ']' )?  ('=' expr  | ch_init)?;
ch_init  :  ID ID;
varref  : ID ( '[' expr ']' )? ( '.' varref )?;
```

```
send    : varref '!' varref '('(arg_lst)?')';
receive : varref '?' varref ;
arg_lst  : expr ( ',' expr ) *;
assign  : varref '=' expr
        | varref '+' '+'
        | varref '-' '-';
stmnt
        :'if' option 'fi'
          |  'do' option 'od'
          | 'atomic' '{' sequence '}'
          | '{' sequence '}'
          | send
          | receive
          | assign
          | expr
        ;
option  :  ':' ':' sequence ( ':' ':' sequence ) *;
andor  : '&' '&' | '|' '|';
binarop: '+' | '-' | '*' | '/' |
            | '>' | '<' | '>' '=' | '<' '=' | '=' '=' | '!' '=' | andor;
unarop: '~' | '-' | '!';
expr    :  exprbinaryor;
exprbinaryor : (exprbinaryand) ('||' exprbinaryand)*;
exprbinaryand: (exprbinaryeq  ) ('&&' exprbinaryeq  )*;
 exprbinaryeq  :
    (exprbinaryineq) ('='"=' exprbinaryineq | '!"=' exprbinaryineq)*;
exprbinaryineq          : (exprbinaryadd) ('<' exprbinaryadd
        | '>' exprbinaryadd |  '<"=' exprbinaryadd | '>"=' exprbinaryadd)*;
exprbinaryadd :
    (exprbinarymult) ('+' exprbinarymult | '-' exprbinarymult)*;
```

exprbinarymult : (exprsimple) ('' exprsimple | '/' exprsimple)*;*

exprsimple : '(' expr ')' | varref | constant ;

Appendix B - Examples

This section has the code for intermediate generated xml file and the output model for the two examples specified in the CHAPTER – 4.

Example 1 – Token Ring Protocol

IntermediateXMLFile.xml

The following is the code of the intermediate xml file which is generated during the translation process.

```
<?xml version="1.0"?>
<!DOCTYPE IntermediateXmlFileDTD SYSTEM "IntermediateXmlFileDTD.dtd" >
<Application><MessagesCollection>
<Message Type="Token">
</Message>
<Message Type="Request">
</Message></MessagesCollection>
<ChannelsCollection>
<Channel Name="channel12" SourceProcess="node1"
DestinationProcess="node2"></Channel>
<Channel Name="channel14" SourceProcess="node1"
DestinationProcess="node4"></Channel>
<Channel Name="channel21" SourceProcess="node2"
DestinationProcess="node1"></Channel>
<Channel Name="channel23" SourceProcess="node2"
DestinationProcess="node3"></Channel>
<Channel Name="channel34" SourceProcess="node3"
DestinationProcess="node4"></Channel>
<Channel Name="channel32" SourceProcess="node3"
DestinationProcess="node2"></Channel>
```

<Channel Name="channel41" SourceProcess="node4"

DestinationProcess="node1"></Channel>

<Channel Name="channel43" SourceProcess="node4"

DestinationProcess="node3"></Channel></ChannelsCollection>

<ProcessesCollection>

<Process Name="node1" ProcessID="1" AlgorithmName="node1">

<LocalVariablesCollection>

<LocalVariable Name="hastoken" Value="0" DataType="int"></LocalVariable>

<LocalVariable Name="incs" Value="0" DataType="int"></LocalVariable>

<LocalVariable Name="reqsent" Value="0" DataType="int"></LocalVariable>

<LocalVariable Name="reqreceived" Value="0" DataType="int"></LocalVariable>

<LocalVariable Name="i" Value="0" DataType="int"></LocalVariable>

</LocalVariablesCollection>

<GuardActionCollection>

<GuardAction>

<Guards>

<Guard>hastoken==1&&incs==0&&reqreceived==0&&i<1</Guard>

</Guards>

<Actions>

<Action>incs=1</Action>

<Action>i++</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard>hastoken==1&&incs==0&&reqreceived==1</Guard>

</Guards>

<Actions>

<Action Type="send">channel12!Token()</Action>

<Action>hastoken=0</Action>

<Action>reqreceived=0</Action>

<Action>i=0</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard Type="recieve">channel41?Token</Guard>

</Guards>

<Actions>

<Action>hastoken=1</Action>

<Action>reqsent=0</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard>hastoken==0&& reqsent==0</Guard>

</Guards>

<Actions>

<Action Type="send">channel14!Request()</Action>

<Action>reqsent=1</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard Type="recieve">channel21?Request</Guard>

</Guards>

<Actions>

<Action>reqreceived=1</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>
<Guard>incs==1</Guard>
</Guards>
<Actions>
<Action>incs=0</Action>
</Actions>
</GuardAction>
</GuardActionCollection>
</Process>
<Process Name="node2" ProcessID="2" AlgorithmName="node2">
<LocalVariablesCollection>
<LocalVariable Name="hastoken" Value="1" DataType="int"></LocalVariable>
<LocalVariable Name="incs" Value="0" DataType="int"></LocalVariable>
<LocalVariable Name="reqsent" Value="0" DataType="int"></LocalVariable>
<LocalVariable Name="reqreceived" Value="0" DataType="int"></LocalVariable>
<LocalVariable Name="i" Value="0" DataType="int"></LocalVariable>
</LocalVariablesCollection>
<GuardActionCollection>
<GuardAction>
<Guards>
<Guard>hastoken==1&&incs==0&&reqreceived==0&&i<1</Guard>
</Guards>
<Actions>
<Action>incs=1</Action>
<Action>i++</Action>
</Actions>
</GuardAction>
<GuardAction>
<Guards>
<Guard>hastoken==1&&incs==0&&reqreceived==1</Guard>

</Guards>

<Actions>

<Action Type="send">channel23!Token()</Action>

<Action>hastoken=0</Action>

<Action>reqreceived=0</Action>

<Action>i=0</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard Type="recieve">channel12?Token</Guard>

</Guards>

<Actions>

<Action>hastoken=1</Action>

<Action>reqsent=0</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard>hastoken==0&& reqsent==0</Guard>

</Guards>

<Actions>

<Action Type="send">channel21!Request()</Action>

<Action>reqsent=1</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard Type="recieve">channel32?Request</Guard>

</Guards>

<Actions>

<Action>reqreceived=1</Action>
</Actions>
</GuardAction>
<GuardAction>
<Guards>
<Guard>incs==1</Guard>
</Guards>
<Actions>
<Action>incs=0</Action>
</Actions>
</GuardAction>
</GuardActionCollection>
</Process>
<Process Name="node3" ProcessID="3" AlgorithmName="node3">
<LocalVariablesCollection>
<LocalVariable Name="hastoken" Value="0" DataType="int"></LocalVariable>
<LocalVariable Name="incs" Value="0" DataType="int"></LocalVariable>
<LocalVariable Name="reqsent" Value="0" DataType="int"></LocalVariable>
<LocalVariable Name="reqreceived" Value="0" DataType="int"></LocalVariable>
<LocalVariable Name="i" Value="0" DataType="int"></LocalVariable>
</LocalVariablesCollection>
<GuardActionCollection>
<GuardAction>
<Guards>
<Guard>hastoken==1&&incs==0&&reqreceived==0&&i<1</Guard>
</Guards>
<Actions>
<Action>incs=1</Action>
<Action>i++</Action>
</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard>hastoken==1&incs==0&reqreceived==1</Guard>

</Guards>

<Actions>

<Action Type="send">channel34!Token()</Action>

<Action>hastoken=0</Action>

<Action>reqreceived=0</Action>

<Action>i=0</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard Type="recieve">channel23?Token</Guard>

</Guards>

<Actions>

<Action>hastoken=1</Action>

<Action>reqsent=0</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard>hastoken==0& reqsent==0</Guard>

</Guards>

<Actions>

<Action Type="send">channel32!Request()</Action>

<Action>reqsent=1</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>
<Guard Type="recieve">channel43?Request</Guard>
</Guards>
<Actions>
<Action>reqreceived=1</Action>
</Actions>
</GuardAction>
<GuardAction>
<Guards>
<Guard>incs==1</Guard>
</Guards>
<Actions>
<Action>incs=0</Action>
</Actions>
</GuardAction>
</GuardActionCollection>
</Process>
<Process Name="node4" ProcessID="4" AlgorithmName="node4">
<LocalVariablesCollection>
<LocalVariable Name="hastoken" Value="0" DataType="int"></LocalVariable>
<LocalVariable Name="incs" Value="0" DataType="int"></LocalVariable>
<LocalVariable Name="reqsent" Value="0" DataType="int"></LocalVariable>
<LocalVariable Name="reqreceived" Value="0" DataType="int"></LocalVariable>
<LocalVariable Name="i" Value="0" DataType="int"></LocalVariable>
</LocalVariablesCollection>
<GuardActionCollection>
<GuardAction>
<Guards>
<Guard>hastoken==1&&incs==0&&reqreceived==0&&i<1</Guard>
</Guards>

<Actions>
<Action>incs=1</Action>
<Action>i++</Action>
</Actions>
</GuardAction>
<GuardAction>
<Guards>
<Guard>hastoken==1&&incs==0&&reqreceived==1</Guard>
</Guards>
<Actions>
<Action Type="send">channel41!Token()</Action>
<Action>hastoken=0</Action>
<Action>reqreceived=0</Action>
<Action>i=0</Action>
</Actions>
</GuardAction>
<GuardAction>
<Guards>
<Guard Type="recieve">channel34?Token</Guard>
</Guards>
<Actions>
<Action>hastoken=1</Action>
<Action>reqsent=0</Action>
</Actions>
</GuardAction>
<GuardAction>
<Guards>
<Guard>hastoken==0&& reqsent==0</Guard>
</Guards>
<Actions>
<Action Type="send">channel43!Request()</Action>

<Action>reqsent=1</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard Type="recieve">channel14?Request</Guard>

</Guards>

<Actions>

<Action>reqreceived=1</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard>incs==1</Guard>

</Guards>

<Actions>

<Action>incs=0</Action>

</Actions>

</GuardAction>

</GuardActionCollection>

</Process>

</ProcessesCollection>

</Application>

Output Model

The following is the generated code of the output model.

import daj.;*

import daj.awt.;*

public class Finalprogram extends Application {

public static void main(String[] args) {new Finalprogram().run();}

```
public Finalprogram(String t, int x, int y) {super(t, x, y);}
public Finalprogram() {super(" Finalprogram Application", 500, 500);}
private static final long serialVersionUID = 1L;
public void construct() {
Node node1 = node(new node1(1), "1", 20, 40);
Node node2 = node(new node2(2), "2", 40, 80);
Node node3 = node(new node3(3), "3", 60, 120);
Node node4 = node(new node4(4), "4", 80, 160);
link(node1, node2); link(node1, node4);
link(node2, node1);   link(node2, node3);
link(node3, node4);  link(node3, node2);
link(node4, node1);  link(node4, node3);}
 public String getText() {return " ";}
@Override
public void resetStatistics () {}
class Token extends Message {
String message;
 public Token(String i) {message = i;}
 public String value() {return message;}
 public String getText() {return "Message = " + String.valueOf(message);}
}
class Request extends Message {
  String message;
 public Request(String i) {
 message = i;}
public String value() {return message;}
public String getText() {return "Message = " + String.valueOf(message);}
}
class node1 extends Program {
 Message msg;
int NodeID;
```

```
Token Tokenobj;
Request Requestobj;
 int hastoken;
 int incs;
 int reqsent;
 int reqreceived;
 int i;
public node1(int i) {NodeID = i;
msg = null;
hastoken = 0;
incs = 0;
reqsent = 0;
reqreceived = 0;
i = 0;
}
protected void main() {
while (true) {
if (hastoken == 1 && incs == 0 && reqreceived == 0 && i < 1) {
incs = 1;i++;
}
else if (hastoken == 1 && incs == 0 && reqreceived == 1) {
out(0).send(new Token("Token"));hastoken = 0;
reqreceived = 0; i = 0;}
else if (msg instanceof Token) {
Tokenobj = (Token) msg;       hastoken = 1;
 reqsent = 0; msg = null;       }
else if (hastoken == 0 && reqsent == 0) {
 out(1).send(new Request("Request"));reqsent = 1;}
 else if (msg instanceof Request) {
 Requestobj = (Request) msg; reqreceived = 1;
msg = null;
```

```
}
 else if (incs == 1) {
incs = 0;}
else {
msg = in(in().select()).receive();}
}
}
 public String getText() {
String msgString;
if (msg == null)
msgString = "(null)";
  else
 msgString = msg.getText();
 return "hastoken: " + String.valueOf(hastoken) + "\n" + "incs: "
+ String.valueOf(incs) + "\n" + "reqsent: "
+ String.valueOf(reqsent) + "\n" + "reqreceived: "
+ String.valueOf(reqreceived) + "\n" + "i: "
+ String.valueOf(i) + "\n" + "NodeID:"
+ String.valueOf(this.NodeID);
}
}
 class node2 extends Program {
Message msg;
int NodeID;
Token Tokenobj;
Request Requestobj;
int hastoken;
int incs;
int reqsent;
int reqreceived;
int i;
```

```
public node2(int i) {
NodeID = i;
msg = null;
hastoken = 1;
incs = 0;
reqsent = 0;
reqreceived = 0;
i = 0;
}
protected void main() {
while (true) {
if (hastoken == 1 && incs == 0 && reqreceived == 0 && i < 1) {
incs = 1;i++;
}
else if (hastoken == 1 && incs == 0 && reqreceived == 1) {
out(1).send(new Token("Token"));hastoken = 0;
reqreceived = 0;i = 0;}
 else if (msg instanceof Token) {
Tokenobj = (Token) msg; hastoken = 1;
reqsent = 0; msg = null;}
else if (hastoken == 0 && reqsent == 0) {
out(0).send(new Request("Request"));
reqsent = 1;}
 else if (msg instanceof Request) {
 Requestobj = (Request) msg;
reqreceived = 1;msg = null;}
else if (incs == 1) {
incs = 0;}
 else {
msg = in(in().select()).receive();
}
```

```
}
}
public String getText()
 {
String msgString;
 if (msg == null)
msgString = "(null)";
 else
msgString = msg.getText();
return "hastoken: " + String.valueOf(hastoken) + "\n" + "incs: "
+ String.valueOf(incs) + "\n" + "reqsent: "
+ String.valueOf(reqsent) + "\n" + "reqreceived: "
+ String.valueOf(reqreceived) + "\n" + "i: "
+ String.valueOf(i) + "\n" + "NodeID:"
+ String.valueOf(this.NodeID);
}
}

class node3 extends Program {
Message msg;
int NodeID;
Token Tokenobj;
Request Requestobj;
int hastoken;
int incs;
int reqsent;
int reqreceived;
int i;
public node3(int i) {
NodeID = i;
msg = null;
```

```
hastoken = 0;
incs = 0;
reqsent = 0;
reqreceived = 0;
i = 0;
}

protected void main() {
while (true) {
if (hastoken == 1 && incs == 0 && reqreceived == 0 && i < 1) {
incs = 1;i++;}
else if (hastoken == 1 && incs == 0 && reqreceived == 1) {
out(0).send(new Token("Token"));hastoken = 0;
reqreceived = 0;i = 0;}
  else if (msg instanceof Token) {
 Tokenobj = (Token) msg; hastoken = 1;
 reqsent = 0;msg = null;
 }
  else if (hastoken == 0 && reqsent == 0) {
  out(1).send(new Request("Request"));
  reqsent = 1;
  }
  else if (msg instanceof Request) {
 Requestobj = (Request) msg;
 reqreceived = 1;msg = null;
  }
 else if (incs == 1) {
incs = 0;
}
else {msg = in(in().select()).receive();}
}
```

```
}
public String getText() {
String msgString;
if (msg == null)
msgString = "(null)";
else
msgString = msg.getText();
return "hastoken: " + String.valueOf(hastoken) + "\n" + "incs: "
+ String.valueOf(incs) + "\n" + "reqsent: "
+ String.valueOf(reqsent) + "\n" + "reqreceived: "
+ String.valueOf(reqreceived) + "\n" + "i: "
+ String.valueOf(i) + "\n" + "NodeID:"
+ String.valueOf(this.NodeID);
 }
}
class node4 extends Program {
Message msg;
int NodeID;
Token Tokenobj;
Request Requestobj;
int hastoken;
int incs;
int reqsent;
int reqreceived;
int i;
public node4(int i) {
NodeID = i;
msg = null;
hastoken = 0;incs = 0;
reqsent = 0;
reqreceived = 0;
```

```
 i = 0;
}
 protected void main() {
while (true) {
if (hastoken == 1 && incs == 0 && reqreceived == 0 && i < 1)            {
incs = 1;i++;  }
 else if (hastoken == 1 && incs == 0 && reqreceived == 1) {
out(0).send(new Token("Token"));hastoken = 0;
 reqreceived = 0;i = 0;}
 else if (msg instanceof Token) {
Tokenobj = (Token) msg; hastoken = 1;
reqsent = 0; msg = null;}
else if (hastoken == 0 && reqsent == 0) {
 out(1).send(new Request("Request"));reqsent = 1;}
 else if (msg instanceof Request) {
Requestobj = (Request) msg;
reqreceived = 1;
msg = null;
}
 else if (incs == 1) {
incs = 0;
}
 else {msg = in(in().select()).receive();}}}
public String getText() {
String msgString;
if (msg == null)
msgString = "(null)";
else
msgString = msg.getText();
return "hastoken: " + String.valueOf(hastoken) + "\n" + "incs: "
+ String.valueOf(incs) + "\n" + "reqsent: "
```

```
+ String.valueOf(reqsent) + "\n" + "reqreceived: "
+ String.valueOf(reqreceived) + "\n" + "i: "
+ String.valueOf(i) + "\n" + "NodeID:"
+ String.valueOf(this.NodeID);
}
}
}
```

Example 2 – Six Process in a Tree Topology

IntermediateXMLFile.xml

The following is the code of the intermediate xml file which is generated during the translation process.

```
<?xml version="1.0"?>
<!DOCTYPE IntermediateXmlFileDTD SYSTEM "IntermediateXmlFileDTD.dtd" >
<Application>
<MessagesCollection>
<Message Type="M">
</Message>
<Message Type="ack">
</Message>
<Message Type="ackr">
</Message>
<Message Type="ackv">
</Message>
</MessagesCollection>
<ChannelsCollection>
<Channel Name="channel12" SourceProcess="node1"
DestinationProcess="node2"></Channel>
<Channel Name="channel21" SourceProcess="node2"
DestinationProcess="node1"></Channel>
```

<Channel Name="channel13" SourceProcess="node1"
DestinationProcess="node3"></Channel>
<Channel Name="channel31" SourceProcess="node3"
DestinationProcess="node1"></Channel>
<Channel Name="channel24" SourceProcess="node2"
DestinationProcess="node4"></Channel>
<Channel Name="channel42" SourceProcess="node4"
DestinationProcess="node2"></Channel>
<Channel Name="channel35" SourceProcess="node3"
DestinationProcess="node5"></Channel>
<Channel Name="channel53" SourceProcess="node5"
DestinationProcess="node3"></Channel>
<Channel Name="channel63" SourceProcess="node6"
DestinationProcess="node3"></Channel>
<Channel Name="channel36" SourceProcess="node3"
DestinationProcess="node6"></Channel>
</ChannelsCollection>
<ProcessesCollection>
<Process Name="node1" ProcessID="1" AlgorithmName="node1">
<LocalVariablesCollection>
<LocalVariable Name="flag" Value="1" DataType="int"></LocalVariable>
<LocalVariable Name="ackrecv1" Value="0" DataType="int"></LocalVariable>
<LocalVariable Name="ackrecv2" Value="0" DataType="int"></LocalVariable>
</LocalVariablesCollection>
<GuardActionCollection>
<GuardAction>
<Guards>
<Guard>flag==1</Guard>
</Guards>
<Actions>
<Action Type="send">channel12!M()</Action>

<Action Type="send">channel13!M()</Action>

<Action>flag=0</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard Type="recieve">channel21?ackr</Guard>

</Guards>

<Actions>

<Action>ackrecv1=1</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard Type="recieve">channel31?ackv</Guard>

</Guards>

<Actions>

<Action>ackrecv2=1</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard>ackrecv1==1&&ackrecv2==1</Guard>

</Guards>

<Actions>

<Action>flag=1</Action>

<Action>ackrecv1=0</Action>

<Action>ackrecv2=0</Action>

</Actions>

</GuardAction>

</GuardActionCollection>

</Process>

<Process Name="node2" ProcessID="2" AlgorithmName="node2">

<LocalVariablesCollection>

<LocalVariable Name="flag" Value="0" DataType="int"></LocalVariable>

</LocalVariablesCollection>

<GuardActionCollection>

<GuardAction>

<Guards>

<Guard>flag==1</Guard>

</Guards>

<Actions>

<Action Type="send">channel21!ackr()</Action>

<Action>flag=0</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard Type="recieve">channel12?M</Guard>

</Guards>

<Actions>

<Action>flag=2</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard>flag==2</Guard>

</Guards>

<Actions>

<Action Type="send">channel24!M()</Action>

<Action>flag=3</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard Type="recieve">channel42?ack</Guard>

</Guards>

<Actions>

<Action>flag=1</Action>

</Actions>

</GuardAction>

</GuardActionCollection>

</Process>

<Process Name="node3" ProcessID="3" AlgorithmName="node3">

<LocalVariablesCollection>

<LocalVariable Name="flag" Value="0" DataType="int"></LocalVariable>

<LocalVariable Name="ackrecv1" Value="0" DataType="int"></LocalVariable>

<LocalVariable Name="ackrecv2" Value="0" DataType="int"></LocalVariable>

</LocalVariablesCollection>

<GuardActionCollection>

<GuardAction>

<Guards>

<Guard>ackrecv1==1&&ackrecv2==1</Guard>

</Guards>

<Actions>

<Action>flag=1</Action>

<Action>ackrecv1=0</Action>

<Action>ackrecv2=0</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard>flag==1</Guard>

</Guards>

<Actions>

<Action Type="send">channel31!ackv()</Action>

<Action>flag=0</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard Type="recieve">channel53?ackr</Guard>

</Guards>

<Actions>

<Action>ackrecv1=1</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard Type="recieve">channel63?ackv</Guard>

</Guards>

<Actions>

<Action>ackrecv2=1</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard Type="recieve">channel13?M</Guard>

</Guards>

<Actions>

<Action>flag=2</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>
<Guard>flag==2</Guard>
</Guards>
<Actions>
<Action Type="send">channel35!M()</Action>
<Action Type="send">channel36!M()</Action>
<Action>flag=3</Action>
</Actions>
</GuardAction>
</GuardActionCollection>
</Process>
<Process Name="node4" ProcessID="4" AlgorithmName="node4">
<LocalVariablesCollection>
<LocalVariable Name="flag" Value="0" DataType="int"></LocalVariable>
</LocalVariablesCollection>
<GuardActionCollection>
<GuardAction>
<Guards>
<Guard>flag==1</Guard>
</Guards>
<Actions>
<Action Type="send">channel42!ack()</Action>
<Action>flag=0</Action>
</Actions>
</GuardAction>
<GuardAction>
<Guards>
<Guard Type="recieve">channel24?M</Guard>
</Guards>
<Actions>
<Action>flag=1</Action>

</Actions>

</GuardAction>

</GuardActionCollection>

</Process>

<Process Name="node5" ProcessID="5" AlgorithmName="node5">

<LocalVariablesCollection>

<LocalVariable Name="flag" Value="0" DataType="int"></LocalVariable>

</LocalVariablesCollection>

<GuardActionCollection>

<GuardAction>

<Guards>

<Guard>flag==1</Guard>

</Guards>

<Actions>

<Action Type="send">channel53!ackr()</Action>

<Action>flag=0</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard Type="recieve">channel35?M</Guard>

</Guards>

<Actions>

<Action>flag=1</Action>

</Actions>

</GuardAction>

</GuardActionCollection>

</Process>

<Process Name="node6" ProcessID="6" AlgorithmName="node6">

<LocalVariablesCollection>

<LocalVariable Name="flag" Value="0" DataType="int"></LocalVariable>

</LocalVariablesCollection>

<GuardActionCollection>

<GuardAction>

<Guards>

<Guard>flag==1</Guard>

</Guards>

<Actions>

<Action Type="send">channel63!ackv()</Action>

<Action>flag=0</Action>

</Actions>

</GuardAction>

<GuardAction>

<Guards>

<Guard Type="recieve">channel36?M</Guard>

</Guards>

<Actions>

<Action>flag=1</Action>

</Actions>

</GuardAction>

</GuardActionCollection>

</Process>

</ProcessesCollection>

</Application>

Output Model

The following is the generated code of the output model.

import daj.;*

import daj.awt.;*

public class Finalprogram extends Application {

public static void main(String[] args) {

```
new Finalprogram().run();
public Finalprogram(String t, int x, int y) {
super(t, x, y);
}
public Finalprogram() {
super(" Finalprogram Application", 500, 500);
}
private static final long serialVersionUID = 1L;
public void construct() {
Node node1 = node(new node1(1), "1", 20, 40);
Node node2 = node(new node2(2), "2", 40, 80);
Node node3 = node(new node3(3), "3", 60, 120);
Node node4 = node(new node4(4), "4", 80, 160);
Node node5 = node(new node5(5), "5", 100, 200);
Node node6 = node(new node6(6), "6", 120, 240);
link(node1, node2);
link(node2, node1);
link(node1, node3);
link(node3, node1);
link(node2, node4);
link(node4, node2);
link(node3, node5);
link(node5, node3);
link(node6, node3);
link(node3, node6);
}
public String getText() {
return " ";
}
@Override
public void resetStatistics() {
```

```
}
class M extends Message {
String message;
public M(String i) {
message = i;
}
public String value() {
return message;
}
public String getText() {
return "Message = " + String.valueOf(message);
}
}
class ack extends Message {
String message;
public ack(String i) {
message = i;
}
public String value() {
return message;
}
public String getText() {
return "Message = " + String.valueOf(message);
}
}
 class ackr extends Message {
String message;
public ackr(String i) {
message = i;
}
public String value() {
```

```
return message;
}
public String getText() {
return "Message = " + String.valueOf(message);
}
}
 class ackv extends Message {
 String message;
 public ackv(String i) {
 message = i;
}
 public String value() {
return message;
}
 public String getText() {
return "Message = " + String.valueOf(message);
}
}
 class node1 extends Program {
Message msg;
int NodeID;
M Mobj;
ack ackobj;
ackr ackrobj;
ackv ackvobj;
int flag;
int ackrecv1;
int ackrecv2;
public node1(int i) {
NodeID = i;
msg = null;
```

```
flag = 1;
ackrecv1 = 0;
ackrecv2 = 0;
}
protected void main() {
while (true) {
if (flag == 1) {
out(0).send(new M("M"));
out(1).send(new M("M"));
flag = 0;
} else if (msg instanceof ackr) {
ackrobj = (ackr) msg;
ackrecv1 = 1;
msg = null;
} else if (msg instanceof ackv) {
ackvobj = (ackv) msg;
ackrecv2 = 1;
msg = null;
} else if (ackrecv1 == 1 && ackrecv2 == 1) {
flag = 1;
ackrecv1 = 0;
ackrecv2 = 0;
} else {
msg = in(in().select()).receive();
}
}
}
public String getText() {
String msgString;
if (msg == null)
msgString = "(null)";
```

```
else
msgString = msg.getText();
return "flag: " + String.valueOf(flag) + "\n" + "ackrecv1: "
+ String.valueOf(ackrecv1) + "\n" + "ackrecv2: "
+ String.valueOf(ackrecv2) + "\n" + "NodeID:"
+ String.valueOf(this.NodeID);
}
}
class node2 extends Program {
Message msg;
int NodeID;
M Mobj;
ack ackobj;
ackr ackrobj;
ackv ackvobj;
int flag;
public node2(int i) {
NodeID = i;
msg = null;
flag = 0;
}
protected void main() {
while (true) {
if (flag == 1) {
out(0).send(new ackr("ackr"));
flag = 0;
} else if (msg instanceof M) {
Mobj = (M) msg;
flag = 2;
msg = null;
} else if (flag == 2) {
```

```
out(1).send(new M("M"));
flag = 3;
} else if (msg instanceof ack) {
ackobj = (ack) msg;
flag = 1;
msg = null;
} else {
msg = in(in().select()).receive();
}
}
}
public String getText() {
String msgString;
if (msg == null)
msgString = "(null)";
else
msgString = msg.getText();
return "flag: " + String.valueOf(flag) + "\n" + "NodeID:"
+ String.valueOf(this.NodeID);
}
}
class node3 extends Program {
Message msg;
int NodeID;
M Mobj;
ack ackobj;
ackr ackrobj;
ackv ackvobj;
int flag;
int ackrecv1;
int ackrecv2;
```

```
public node3(int i) {
NodeID = i;
msg = null;
flag = 0;
ackrecv1 = 0;
ackrecv2 = 0;
}
protected void main() {
while (true) {
if (ackrecv1 == 1 && ackrecv2 == 1) {
flag = 1;
ackrecv1 = 0;
ackrecv2 = 0;
} else if (flag == 1) {
out(0).send(new ackv("ackv"));
flag = 0;
} else if (msg instanceof ackr) {
ackrobj = (ackr) msg;
ackrecv1 = 1;
msg = null;
} else if (msg instanceof ackv) {
ackvobj = (ackv) msg;
ackrecv2 = 1;
msg = null;
} else if (msg instanceof M) {
Mobj = (M) msg;
flag = 2;
msg = null;
} else if (flag == 2) {
out(1).send(new M("M"));
out(2).send(new M("M"));
```

```
flag = 3;
} else {
msg = in(in().select()).receive();
}
}
}
public String getText() {
String msgString;
if (msg == null)
msgString = "(null)";
else
msgString = msg.getText();
return "flag: " + String.valueOf(flag) + "\n" + "ackrecv1: "
+ String.valueOf(ackrecv1) + "\n" + "ackrecv2: "
+ String.valueOf(ackrecv2) + "\n" + "NodeID:"
+ String.valueOf(this.NodeID);
}
}
class node4 extends Program {
Message msg;
int NodeID;
M Mobj;
ack ackobj;
ackr ackrobj;
ackv ackvobj;
int flag;
public node4(int i) {
NodeID = i;
msg = null;
flag = 0;
}
```

```
protected void main() {
while (true) {
if (flag == 1) {
out(0).send(new ack("ack"));
flag = 0;
} else if (msg instanceof M) {
Mobj = (M) msg;
flag = 1;
msg = null;
} else {
 msg = in(in().select()).receive();
 }
 }
}
public String getText() {
String msgString;
if (msg == null)
msgString = "(null)";
else
msgString = msg.getText();
return "flag: " + String.valueOf(flag) + "\n" + "NodeID:"
+ String.valueOf(this.NodeID);
}
}
class node5 extends Program {
Message msg;
int NodeID;
M Mobj;
ack ackobj;
ackr ackrobj;
ackv ackvobj;
```

```
int flag;
public node5(int i) {
NodeID = i;
msg = null;
flag = 0;
}
protected void main() {
while (true) {
if (flag == 1) {
out(0).send(new ackr("ackr"));
flag = 0;
} else if (msg instanceof M) {
Mobj = (M) msg;
flag = 1;
msg = null;
} else {
msg = in(in().select()).receive();
}
}
}
public String getText() {
String msgString;
if (msg == null)
msgString = "(null)";
else
msgString = msg.getText();
return "flag: " + String.valueOf(flag) + "\n" + "NodeID:"
+ String.valueOf(this.NodeID);
}
}
class node6 extends Program {
```

```
Message msg;
int NodeID;
M Mobj;
ack ackobj;
ackr ackrobj;
ackv ackvobj;
int flag;
public node6(int i) {
NodeID = i;
msg = null;
flag = 0;
}
protected void main() {
while (true) {
if (flag == 1) {
out(0).send(new ackv("ackv"));
flag = 0;
} else if (msg instanceof M) {
Mobj = (M) msg;
flag = 1;
msg = null;
} else {
msg = in(in().select()).receive();
}
}
}
public String getText() {
String msgString;
if (msg == null)
 msgString = "(null)";
else
```

```
 msgString = msg.getText();
return "flag: " + String.valueOf(flag) + "\n" + "NodeID:"
+ String.valueOf(this.NodeID);
}
}
}
```